NVIDIA Audio Effects SDK

Programming Guide

PG-09731-001_v1.0 | April 2021

Document History

PG-09731-001_v1.0

Version	Date	Description of Change
v0.1 Alpha	January 2020	Alpha release
v0.2 Beta	March 2020	Beta release
v0.5 Beta	October 2020	Beta release
v0.6 Beta	January 2021	Here are the updates in this release: ▶ Added GPU selection support (page 12). ▶ Updated the noise profile list (page 6). ▶ Added `GetSupportedDevices()` API documentation (page 10). ▶ Added `NvAFX_GetSupportedDevices()` API reference (page 23).
v1.0	April 2021	Here are the updates in this release: ▶ Modified name of sample app from `denoise_wav` to `effects_demo` (page 3 and 4). ▶ Added information about the suppression and cancellation effects (page 6, 7 and 8). ▶ Modified information about NvAFX_Run (page 24). ▶ Modified some functions (page 16). ▶ Modified some type definitions (page 15) ▶ Updated spell checks (page 9) ▶ Modified information (page 8) ▶ Minor change (page 20) ▶ Updated latency numbers (page 9)

Table of Contents

Chapter 1. Introduction to NVIDIA Audio Effects SDK

NVIDIA® Audio Effects SDK is used to apply effects to audio. The SDK is powered by NVIDIA RTX™ graphic processor units (GPUs) with Tensor Cores, so the algorithm throughput is greatly accelerated, and latency is reduced. Refer to Tensor Cores for more information. By leveraging the capabilities of NVIDIA RTX GPUs, developers can use the SDK to build audio plugins and add sound effects for broadcasting.

NVIDIA Audio Effects SDK provides the following audio effects for broadcast use cases with real-time audio processing.

- **Denoising**: Recordings of speech made outside of a recording studio can contain a lot of background noise, which causes the speech to be garbled and difficult to understand.

  The audio denoising effect removes the background noise.

- **Dereverb/Room echo cancellation**: Recordings of speech might contain the reverberation from the recording environment.

  Excessive reverb reduces speech clarity, and the dereverb effect helps remove or suppress the reverb.

- **Denoise plus Dereverb:** The effects listed above are combined to remove/suppress the noise and reverb.

Note: In this guide, the term *Room Echo Cancellation* is used interchangeably with *Dereverb*.

Chapter 2. Getting Started with NVIDIA Audio Effects SDK

2.1 Hardware and Software Requirements

NVIDIA Audio Effects SDK requires specific GPUs and a specific version of the Windows OS and other associated software on which the SDK depends.

2.1.1 Hardware Requirements

The SDK is supported on NVIDIA GPUs with Tensor Cores.

Hardware	Required Version
NVIDIA GPU	NVIDIA GPUs with Tensor Cores

2.1.1 Software Requirements

NVIDIA Audio Effects SDK requires a specific version of the Windows OS and other associated software on which the SDK depends. The NVIDIA CUDA® and TensorRT™ dependencies are bundled with the SDK Installer. See "Installing the NVIDIA Audio Effects SDK" on page 3.

NVIDIA Audio Effects SDK is designed and optimized for client-side application integration and for local deployment. We **do not** officially support the testing, experimentation, deployment of this SDK in a datacenter/cloud environment.

Software	Required Version
Windows OS	64-bit Windows 10
Microsoft Visual Studio	2015 (MSVC14.0) or later
CMake	3.9 or later
NVIDIA Graphics Driver for Windows	460.34 or later

2.2 Installing the NVIDIA Audio Effects SDK

NVIDIA Audio Effects SDK is distributed in the following parts:

- A *developer package* that contains the AI models, binaries, header file, and a sample app.
- A *redistributable package* that contains only the AI models and binaries.

  This package streamlines the installation and usage of the SDK on the end-user's computer.

To develop applications with the NVIDIA Audio Effects SDK, you must install the *developer package* and provide the path to this software during compilation and linking. The app will use the SDK functions that are exposed by the SDK header and dynamically link against the provided libraries.

On the end-user's computer, when the *redistributable package* is installed, the installer completes the following tasks:

- Copies the AI models and binaries to the install location.
- Sets the `NVAFX_SDK_DIR` environmental variable that points to the directory where the *redistributable package* is installed and that contains the AI models and binaries.

  The app needs to use this variable to dynamically link and load the binaries and the AI model.

2.3 NVIDIA Audio Effects SDK Sample Application

To demonstrate the various audio effects, the SDK provides the following options:

- The sample includes the source file `effects_demo.cpp` that can be compiled and run.
- The sample application is also available as a binary file `effects_demo.exe` that can be executed directly.

2.3.1 Building the Sample Application

The SDK includes the source code for building the sample application.

1. Start the CMake GUI and specify the source folder and a build folder for the binary files.
   a). For the source folder, ensure that the path ends in `package`.
   b). For the build folder, ensure that the path ends in `package/build`.
2. Use CMake to configure and generate the Visual Studio solution file.
   a). Click **Configure**.
   b). When prompted to confirm whether CMake can create the build folder, click **OK**.
   c). To enable CMake to locate the CUDA complier, select **Visual Studio** for the generator and **x64** for the platform.
   d). To finish configuring the Visual Studio solution file, click **Finish**.
   e). To generate the Visual Studio solution file, click **Generate**.
3. Use Visual Studio to generate the application binary (`.exe`) file from the solution file that was generated in the previous step.
   a). In CMake, click **Open Project** to open Visual Studio.
   b). In Visual Studio, select **Build** > **Build Solution**.

2.3.2 Running the Sample Application

In a Command Prompt window, enter the following command:

```
effects_demo.exe -c config-file
```

-c config-file

Specifies the path of the effect sample config file, for example, `denoise48k_cfg_turing.txt`. A few sample config files are supplied with the sample application.

The following example runs the `effects_demo.exe` sample application:

```
effects_demo.exe -c denoise48k_cfg_turing.txt
```

The config files contain the following parameters with one pair per line:

effect "effect to be applied"

Specifies the effect which needs to be applied, for example, denoiser. Currently, denoiser, dereverb, and dereverb+denoiser are the supported effects.

sample_rate audio-sample-rate

Specifies the sample rate of the audio, for example, 48000. Currently, the 16 kHz and 48 kHz rates are supported.

model model-file

Specifies the path of the model file that will be used in the sample application, for example, `denoiser_48k.trtpkg`. The model file should match the audio sample rate that was specified in the `sample_rate` parameter.

Note: A 48kHz model file and a 16kHz model file is included with the SDK for all effects.

input_wav input-audio-file

Specifies the path of the noisy input audio `.wav` file to use, for example, `noisy_48k.wav`.

Note: A sample input audio file is included with the sample application.

output_wav output-audio-file

Specifies the path of the file to which the applied effect audio output is to be written, for example, `denoised_48k.wav`. The audio format of the output file will match the audio format of the input file.

Note: Only the .wav file format is supported.

intensity_ratio intensity-ratio

Specifies the denoising intensity ratio. The value of this parameter ranges from `0.0f` to `1.0f`, where a higher value indicates a stronger suppression of noise/reverb. A value of `0.0f` is equivalent to a passthrough.

Chapter 3. Using NVIDIA Audio Effects SDK in Applications

By using the NVIDIA Audio Effects SDK, you can enable an application to apply effects to audio. The NVIDIA Audio Effects API is a C API but can also be used with applications that are built using C++.

3.1 About the Background Noise Suppression Effect

Note: In this guide, the term *Background Noise Suppression* is used interchangeably with *Denoising*.

Recordings of speech made outside of a recording studio contain a lot of background noise. The Audio Denoiser Effect removes the following types of background noise from audio recordings:

- AC noise
- PC noise
- Babble / crowd noise
- Chatter from other people
- Keyboard
- Fan noise
- Sirens
- Clapping
- Tapping
- Sounds of furniture moving
- Sounds of glass breaking
- Traffic noise
- Mouse clicks

- Sounds of a train passing by
- Sounds of a vacuum cleaner
- Washing machine
- Metal sounds
- Baby crying
- Wrappers (plastic / non- plastic rustling)
- Water taps / running water
- Cooking sounds (cutting, cooker, etc)
- Construction site sounds
- Rains
- Pet sounds
- Drums
- Door slamming
- Bird chirping
- Phone ringing

Here is a list of the Audio Denoiser Effect characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.
- The minimum latency is 74 ms.

3.2 About the Room Echo Cancellation Effect

Note: In this guide, the term *Room Echo Cancellation* is used interchangeably with *Dereverb*.

The Room Echo Cancellation feature is in **Beta**.

Recordings of speech made in some large room/hall contains echoes and reverbs. The Audio Room Echo Cancellation Effect removes/suppresses the echoes and reverbs from audio recordings:

Here is a list of the Audio Room Echo Cancellation Effect characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.
- The minimum latency is 74 ms.

3.3 About the Room Echo Cancellation + Background Noise Suppression Effect

Note: In this guide, the term *Room Echo Cancellation + Background Noise Suppression* is used interchangeably with *Dereverb+Denoiser*.

The Room Echo Cancellation + Background Noise Suppression Effect feature is in **Beta**.

This effect combines the denoiser effect (see "About the Background Noise Suppression Effect" on page 6) and Dereverb effect (see "About the Room Echo Cancellation Effect" on page 7) into one effect. It applies both denoising and dereverb effect on the input audio.

Here is a list of the Audio Room Echo Cancellation Effect + Background Noise Removal characteristics:

- The audio format is 48kHz and 16KHz sample rate and 32-bit float type.
- The minimum latency is 74 ms.

3.4 Creating an Audio Effect

Call the `NvAFX_CreateEffect()` function and specify the following information as parameters:

- The `NvAFX_EffectSelector` type `NVAFX_EFFECT_DENOISER`.
- The location where to store the handle to the newly created audio effect.

The `NvAFX_CreateEffect()` function creates a handle to the audio effect instance for use in additional API calls.

This example creates a denoiser audio effect:

```
NvAFX_Status err = NvAFX_CreateEffect(NVAFX_EFFECT_DENOISER, &handle);
```

3.5 Setting the Sample Rate and Path to the Model

An audio effect requires a model to transform the input audio, and each model supports a specific audio sample rate. You must set the input audio sample rate and the path to the model file that will be used and that supports this sample rate.

To set the sample rate, call the `NvAFX_SetU32()` function and specify the following information as parameters:

- The effect handle that was created.
  See "Creating an Audio Effect" on page 8 for more information.
- The selector string `NVAFX_PARAM_SAMPLE_RATE.`
- An unsigned integer value that specifies the sample rate of the audio.

Call the `NvAFX_SetString()` function and specify the following information as parameters:

- The effect handle that was created.
  See "Creating an Audio Effect" on page 8 for more information.
- The selector string `NVAFX_PARAM_MODEL_PATH.`
- A null-terminated string that indicates the path to the model file.

This example sets the sample rate to `sample_rate` and the path to the model that was specified by the `model_file.c_str()` custom function.

```
NvAFX_Status err;
err = NvAFX_SetU32(handle, NVAFX_PARAM_SAMPLE_RATE, sample_rate);
err = NvAFX_SetString(handle, NVAFX_PARAM_MODEL_PATH, model_file.c_str());
```

3.6 Getting the Parameters of an Effect

The number of samples per frame and number of I/O audio channels are preset for the Audio Effect and cannot be changed. Before running an audio effect, you must get the number of samples per frame and the number of I/O channels to pass as parameters to the function. See "Running an Audio Effect" on page 11 for more information.

Note: To ensure that the sample rate of the audio that you are transforming is compatible with the Audio Effect, you can also get the sample rate.

To get one of these parameters, call the `NvAFX_GetU32()` function and specify the following information as parameters:

- The effect handle that was created.
  See "Creating an Audio Effect" on page 8 for more information.
- The selector string for the parameter that you want to get:
  - To get the number of samples per frame, specify `NVAFX_PARAM_NUM_SAMPLES_PER_FRAME`.
  - To get the number of I/O audio channels, specify `NVAFX_PARAM_NUM_CHANNELS`.
  - To get the sample rate, specify `NVAFX_PARAM_SAMPLE_RATE`.
- A pointer to a location where to store the value that you want to get.

This example gets the number of samples per frame, number of I/O channels, and sample rate for an Audio Effect.

```
unsigned num_samples_per_frame, num_channels, sample_rate;
NvAFX_Status err;
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_SAMPLES_PER_FRAME,
&num_samples_per_frame);
err = NvAFX_GetU32(handle, NVAFX_PARAM_NUM_CHANNELS, &num_channels);
err = NvAFX_GetU32(handle, NVAFX_PARAM_SAMPLE_RATE, &sample_rate);
```

3.7 Getting Supported Devices

To run the effect, call the `NvAFX_GetSupportedDevices()` function to fetch the number of supported GPUs, by model.

Note: This method must be called **after** you set the model path.

Here is a list of the parameters:

- The effect handle that was created.
  See "Creating an Audio Effect" on page 8 for more information.
- The size of the input array.
  If the call succeeds, this value will be set by the function.
- Array of size num.
  The function will fill the array with the CUDA device indices of devices that are supported by the model, in descending order of preference, where the first device is the most preferred device.

This example fetches the list of supported GPUs by the model:

```
int numSupportedDevices = 0;
NvAFX_GetSupportedDevices(handle, &numSupportedDevices, nullptr);

std::vector<int> ret(num);
NvAFX_GetSupportedDevices(handle, &numSupportedDevices, ret.data());
```

3.8 Loading an Audio Effect

Loading an effect selects and loads a model and validates the parameters that were set for the effect.

To load an audio effect, call the `NvAFX_Load()` function and specify the effect handle that was created. See "Creating an Audio Effect" on page 8 for more information.

```
NvAFX_Status err = NvAFX_Load(handle);
```

3.9 Running an Audio Effect

After loading an audio effect, run the effect to apply the desired effect. After an effect is run, the contents of the input memory buffer are read, the audio effect is applied, and the output is written to the output memory buffer.

To run an audio effect, call the `NvAFX_Run()` function and pass the following information as parameters:

- The effect handle that was created.
  See "Creating an Audio Effect" on page 8 for more information.
- The input memory buffer to be read.
- The output memory buffer to which the information will be written.
- The number of samples per frame that were obtained.
  See "Getting the Parameters of an Effect" on page 9 for more information.
- The number of I/O audio channels that were obtained.
  See "Getting the Parameters of an Effect" on page 9 for more information.

This example runs an audio effect:

```
NvAFX_Status err = NvAFX_Run(effect, input, output, num_samples,
num_channels);
```

3.10 Destroying an Audio Effect

When an audio effect is no longer required, destroy it to free the resources and the memory that were allocated for the effect.

To destroy an audio effect, call the `NvAFX_DestroyEffect()` function and specify the effect handle that was created. See "Creating an Audio Effect" on page 8 for more information.

```
NvAFX_Status err = NvAFX_DestroyEffect(handle);
```

3.11 Using Multiple GPUs

Applications that are developed with the NVIDIA Audio Effects SDK can be used with multiple GPUs. By default, the SDK assumes that the application will set the GPU. The SDK can optionally select the best GPU to run the effect(s).

3.11.1 Selecting GPU for Audio Effects Processing in a Multi-GPU Environment

The GPU that will be used to run audio effect(s) in a multi-GPU environment can be controlled by using the `cudaSetDevice()` and `cudaGetDevice()` CUDA functions.

Note: The device should be set before calling NvAFX_Load().

The SDK determines the GPU selection based on the compute capability of the currently selected GPU:

- If the currently selected GPU supports the SDK, the effect is loaded.
- If the currently selected GPU does not support the SDK, `NvAFX_Load()` fails.

```
int chosenGPU = 0; // or whatever GPU you want to use
cudaSetDevice(chosenGPU);
NvAFX_Handle effect;
err = NvAFX_API NvAFX_CreateEffect(code, &effect);
err = NvAFX_Set...; // set parameters
…
err = NvAFX_API NvAFX_Load(effect);
…
err = NvAFX_API NvAFX_Run(effect, ...);
```

3.11.2 Offloading GPU Selection to SDK for Audio Effects Processing in a Multi-GPU Environment

To allow the SDK to select the GPU on which to run the audio effect(s), you can use the Audio Effects `NvAFX_SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1)` set function. The `NvAFX_SetU32(NVAFX_PARAM_USE_DEFAULT_GPU)` call is optional and should be called only once **before** an effect is loaded. If it this call is invoked after an audio effect is loaded, it will not have any effect.

If the application sets `NVAFX_PARAM_USE_DEFAULT_GPU` to 1, we assume that the application will not call `cudaSetDevice()`, and all other effects or multiple instances of an effect will use the default GPU selection. If the application calls `cudaSetDevice()` explicitly **before** `NvAFX_Load()`, the SDK will override application's device preference. If the client calls `cudaSetDevice()` to select to a different GPU **before** calling `NvAFX_Run()`, the call will fail.

```
NvAFX_Handle effect;
err = NvAFX_API NvAFX_CreateEffect(code, &effect);
err = NvAFX_API SetU32(effect, NVAFX_PARAM_USE_DEFAULT_GPU, 1);
…
err = NvAFX_API NvAFX_Load(effect);
…
```

3.11.3 Selecting Different GPUs for Different Tasks

In addition to applying the audio effects, your application might be designed to complete multiple tasks in a multi-GPU environment. In this situation, you need to select the best GPU for each task **before** calling `NvAFX_Load()`.

Note: For performance concerns, switching to the appropriate GPU is the responsibility of the application.

If the application does not switch to appropriate GPU before calling `NvAFX_Run()`, the call will fail.

1. To determine the number of GPUs in your environment, call `cudaGetDeviceCount()`.

```
// Get the number of GPUs
cuErr = cudaGetDeviceCount(&deviceCount);
```

2. To get the properties of each GPU and determine which GPU is the best GPU for each task, complete the following operations for each GPU in a loop:
   a). To set the current GPU, call `cudaSetDevice()`.
   b). To get the properties of the current GPU, call `cudaGetDeviceProperties()`.
   c). To determine whether the GPU is the best GPU for each specific task, use the custom code in your application to analyze the properties that are retrieved by `cudaGetDeviceProperties()`.

   This example uses the compute capability, which determines whether a GPU's properties should be analyzed and determines whether the GPU is the best GPU to apply an audio effect filter.

```
// Loop through the GPUs to get the properties of each GPU and
// determine if it is the best GPU for each task based on the
// properties obtained.
for (int dev = 0; dev < deviceCount; ++dev) {
cudaSetDevice(dev);
cudaGetDeviceProperties(&deviceProp, dev);
if (DeviceIsBestForAFX(&deviceProp))  gpuAFX = dev; // say 7.5 compute
if (DeviceIsBestForOtherTask(&deviceProp)) gpuOtherTask = dev;
...
}
cudaSetDevice(gpuAFX);
err = NvAFX_Set...; // set parameters

err = NvAFX_Load(effect, ...);
```

3. To select the GPU for the task **before** performing the task, in the loop to complete the application's tasks, call `cudaSetDevice()`.

```
// Select the best GPU for each task and perform the task.
while (!done) {
  ...
  cudaSetDevice(gpuOtherTask);
  PerformOtherTask();
  cudaSetDevice(gpuAFX);
  err = NvAFX_Run(eff, ...);
```

Chapter 4. NVIDIA Audio Effects SDK API Reference

4.1 Type Definitions

NVIDIA Audio Effects SDK type definitions provide selector strings for the audio effect and the parameters of an audio effect.

4.1.1 NvAFX_EffectSelector

```
typedef const char* NvAFX_EffectSelector;
```

This type definition provides selector strings for the various types of audio effect.

NVAFX_EFFECT_DENOISER: "denoiser"

Denoiser audio effect.

NVAFX_EFFECT_DEREVERB: "dereverb"

Dereverb audio effect.

NVAFX_EFFECT_DEREVERB_DENOISER: "dereverb+denoiser"

Dereverb+Denoiser audio effect.

4.1.2 NvAFX_ParameterSelector

```
typedef const char* NvAFX_ParameterSelector;
```

This type definition provides selector strings for the parameters of an audio effect.

NVAFX_PARAM_MODEL_PATH: "model_path"

A character string that specifies the path to the model file for the effect.

NVAFX_PARAM_SAMPLE_RATE: "sample_rate"

An unsigned integer that specifies the audio sample rate for the effect.

NVAFX_PARAM_NUM_SAMPLES_PER_FRAME: "num_samples_per_frame"

An unsigned integer that specifies the number of samples per frame for the effect.

NVAFX_PARAM_NUM_CHANNELS: "num_channels"

An unsigned integer that specifies the number of I/O audio channels for the effect.

NVAFX_PARAM_INTENSITY_RATIO: "intensity_ratio"

A float value that specifies the factor that ranges from 0.0 to 1.0. Setting the factor to 0.0 is identical to a pass through, and a value of 1.0 provides the maximum possible impact of the effect.

4.1.3 NvAFX_Handle

```
typedef void* NvAFX_Handle;
```

This structure represents the opaque handle that is associated with each instance of an audio effect. Most audio effect function calls include this handle as the first parameter.

4.2 Functions

4.2.1 NvAFX_GetEffectList

```
NvAFX_Status NvAFX_GetEffectList(
  int* num_effects,
  NvAFX_EffectSelector* effects[]
);
```

4.2.1.1 Parameters

num_effects [out]

Type: int*

Address of the buffer that contains the number of effects that are returned in the effects array.

effects [out]

Type: NvAFX_EffectSelector* []

Address to a list of effect selection strings that are supported by the SDK. The list is statically allocated by the API implementation, so the caller does not need to allocate. See "NvAFX_EffectSelector" on page 15 for more information about the selection strings.

4.2.1.2 Return Value

NVAFX_STATUS_SUCCESS on success.

4.2.1.3 Remarks

This function retrieves the list of audio effects that are supported by the SDK. The selection strings for the Audio Effects SDK are populated in the `effects` out parameter. The number of available effects are written to the `num_effects` out parameter.

4.2.2 NvAFX_CreateEffect

```
NvAFX_Status NvAFX_CreateEffect(
  NvAFX_EffectSelector code,
  NvAFX_Handle* effect
);
```

4.2.2.1 Parameters

code [in]

Type: `NvAFX_EffectSelector`

The selection string for the type of audio effect that will be created. See "NvAFX_EffectSelector" on page 15 for more information about the allowed selection strings.

effect [out]

Type: `NvAFX_Handle*`

The location where to store the handle to the newly created audio effect instance.

4.2.2.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.2.3 Remarks

This function creates an instance of the specified type of audio effect and also writes a handle to the audio effect instance to the `effect` out parameter.

4.2.3 NvAFX_DestroyEffect

```
NvAFX_Status NvAFX_DestroyEffect(
  NvAFX_Handle effect
);
```

4.2.3.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance that will be destroyed.

4.2.3.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.3.3 Remarks

This function destroys the audio effect instance with the specified handle and frees resources and memory that were allocated to the instance.

4.2.4 NvAFX_SetString

```
NvAFX_Status NvAFX_SetString(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  const char* val
);
```

4.2.4.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance for which you want to set the specified character string parameter.

param_name [in]

Type: `NvAFX_ParameterSelector`

The selector string `NVAFX_PARAM_MODEL_PATH`.

Any other selector string returns an error.

val [in]

Type: `char*`

Pointer to the character string to which you want to set the parameter.

4.2.4.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.4.3 Remarks

This function sets the value of the specified character string parameter for the specified audio effect to the `val` parameter.

4.2.5 NvAFX_SetU32

```
NvAFX_Status NvAFX_SetU32(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  unsigned int val
);
```

4.2.5.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance for which you want to set the specified character string parameter.

Param_name [in]

Type: `NvAFX_ParameterSelector`

The selector string `NVAFX_PARAM_SAMPLE_RATE`.

Any other selector string returns an error.

val [in]

Type: `unsigned int`

Value to be set for the parameter.

4.2.5.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.5.3 Remarks

This function sets the value of the specified 32-bit unsigned integer parameter for the specified audio effect to the `val` parameter.

4.2.6 NvAFX_SetFloat

```
NvAFX_Status NvAFX_SetFloat(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  float val
);
```

4.2.6.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance for which you want to set the specified float parameter.

Param_name [in]

Type: `NvAFX_ParameterSelector`

The selector string `NVAFX_PARAM_INTENSITY_RATIO`.

Any other selector string returns an error.

val [in]

Type: `float`

Value to be set for the parameter.

4.2.6.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.6.3 Remarks

This function sets the value of the specified float parameter for the specified audio effect to the `val` parameter.

4.2.7 NvAFX_GetString

```
NvAFX_Status NvAFX_GetString(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  char* val,
  int max_length
);
```

4.2.7.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance from which you want to get the specified character string parameter.

Param_name [in]

Type: `NvAFX_ParameterSelector`

The selector string `NVAFX_PARAM_MODEL_PATH.`

Any other selector string returns an error.

val [out]

Type: `char*`

The address of the buffer where the requested character string will be stored.

max_length [in]

Type: `int`

The length in bytes of the buffer that is specified by the `val` parameter.

4.2.7.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.7.3 Remarks

This function gets the value of the character string parameter for the specified audio effect and writes the retrieved string to the buffer at the location specified by the `val` parameter.

4.2.8 NvAFX_GetU32

```
NvAFX_Status NvAFX_GetU32(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  unsigned int* val
);
```

4.2.8.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance from which you want to get the specified 32-bit unsigned integer parameter.

param_name [in]

Type: NvAFX_ParameterSelector

One of the following selector strings for the specified 32-bit unsigned integer parameter that you want to get:

- NVAFX_PARAM_NUM_SAMPLES_PER_FRAME
- NVAFX_PARAM_NUM_CHANNELS
- NVAFX_PARAM_SAMPLE_RATE

Any other selector string returns an error.

Note: NVAFX_PARAM_NUM_SAMPLES_PER_FRAME and NVAFX_PARAM_NUM_CHANNELS parameters are preset for all Effects and cannot be changed. If you call NvAFX_SetU32() to set any of these parameters, the function call returns an error.

val [out]

Type: unsigned int*

The address of the buffer in which the retrieved 32-bit unsigned integer parameter value will be written.

4.2.8.2 Return Value

NVAFX_STATUS_SUCCESS on success.

4.2.8.3 Remarks

This function gets the value of the specified 32-bit unsigned integer parameter for the specified audio effect and writes the retrieved value to the buffer that is specified by the val parameter.

4.2.9 NvAFX_GetFloat

```
NvAFX_Status NvAFX_GetFloat(
  NvAFX_Handle effect,
  NvAFX_ParameterSelector param_name,
  float* val
);
```

4.2.9.1 Parameters

effect [in]

Type: NvAFX_Handle

The handle to the audio effect instance from which you want to get the specified float parameter.

param_name [in]

Type: NvAFX_ParameterSelector

One of the following selector strings for the specified float parameter that you want to get:
NVAFX_PARAM_INTENSITY_RATIO

Any other selector string returns an error.

val [out]

Type: float*

The address of the buffer in which the retrieved float parameter value will be written.

4.2.9.2 Return Value

NVAFX_STATUS_SUCCESS on success.

4.2.9.3 Remarks

This function gets the value of the specified float parameter for the specified audio effect and writes the retrieved value to the buffer that is specified by the val parameter.

4.2.10 NvAFX_GetSupportedDevices

```
NvAFX_Status NvAFX_GetSupportedDevices(
  NvAFX_Handle effect,
  int *num,
  int *devices
);
```

4.2.10.1 Parameters

effect [in]

Type: NvAFX_Handle

The handle to the audio effect instance to load.

num [in, out]

Type: int*

The size of the input array. If call succeeds, this value will be set by the function.

devices [in, out]

Type: int*

Array of size num. The function will fill the array with CUDA device indices of devices supported by the model, in descending order of preference, where the first device is the most preferred device.

4.2.10.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.10.3 Remarks

This function gets the devices supported by the model.

4.2.11 NvAFX_Load

```
NvAFX_Status NvAFX_Load(
  NvAFX_Handle effect
);
```

4.2.11.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance to load.

4.2.11.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.11.3 Remarks

This function loads the specified audio effect and validates the parameters that are set for the effect.

4.2.12 NvAFX_Run

```
NvAFX_Status NvAFX_Run(
  NvAFX_Handle effect,
  const float** input,
  float** output,
  unsigned num_samples,
  unsigned num_channels
);
```

4.2.12.1 Parameters

effect [in]

Type: `NvAFX_Handle`

The handle to the audio effect instance to run.

input [in]

Type: `const float**`

Pointer to an array of buffers where each buffer holds the audio data for one channel. The size of the array must be equal to the number of I/O channels that were preset for the effect. For example, for the Audio Effect, the number of I/O channels must be equal to the value of the `NVAFX_PARAM_NUM_CHANNELS` parameter that was obtained by the `NvAFX_GetU32()` function.

The sample rate of the audio data must be equal to the sample rate that was preset for the effect. For example, for the Audio Effect, the sample rate must be equal to the value of the `NVAFX_PARAM_SAMPLE_RATE` parameter that was obtained by the `NvAFX_GetU32()` function.

output [out]

Type: `float**`

Pointer to an array of buffers to which the output of the effect will be written. After this function returns, each buffer will contain audio data for one channel.

Note: The buffers must already be allocated by the calling program.

- The following macros have been deprecated:
  - `NVAFX_PARAM_DENOISER_SAMPLE_RATE`
  - `NVAFX_PARAM_DENOISER_NUM_CHANNELS`
  - `NVAFX_PARAM_DENOISER_MODEL_PATH`
  - `NVAFX_PARAM_DENOISER_INTENSITY_RATIO`
  - `NVAFX_PARAM_DENOISER_NUM_SAMPLES_PER_FRAME`
- Use the following macros instead:
  - `NVAFX_PARAM_SAMPLE_RATE`
  - `NVAFX_PARAM_NUM_CHANNELS`
  - `NVAFX_PARAM_MODEL_PATH`
  - `NVAFX_PARAM_INTENSITY_RATIO`
  - `NVAFX_PARAM_NUM_SAMPLES_PER_FRAME`

The size of each buffer is same as the size of each buffer that was specified by the `input` parameter.

num_samples [in]

Type: `unsigned`

The number of samples in the input buffer. After this function returns, the buffer that was specified by the `output` parameter will contain the number of samples that were specified in this parameter.

num_channels [in]

Type: `unsigned`

The number of I/O channels.

4.2.12.2 Return Value

`NVAFX_STATUS_SUCCESS` on success.

4.2.12.3 Remarks

This function runs the specified audio effect by reading the contents of the input buffer, applying the audio effect, and writing the output to the output buffer.

effect [in]

paramName [in]

val [in]

4.3 Return Codes

The `NvAFX_Status` enumeration defines the following values that the NVIDIA Audio Effects functions might return to indicate error or success:

NVAFX_STATUS_SUCCESS

Successful execution.

NVAFX_STATUS_FAILED

Generic error code, which indicates that the function failed to execute for an unspecified reason.

NVAFX_STATUS_INVALID_HANDLE

An invalid effect handle has been supplied.

NVAFX_STATUS_INVALID_PARAM

An invalid parameter value has been supplied for this combination of effect and selector string.

NVAFX_STATUS_IMMUTABLE_PARAM

User tried to modify an immutable parameter.

NVAFX_STATUS_INSUFFICIENT_DATA

There is insufficient data to process.

NVAFX_STATUS_EFFECT_NOT_AVAILABLE

The specified effect is not supported.

NVAFX_STATUS_OUTPUT_BUFFER_TOO_SMALL

The output buffer length is too small to hold the requested data.

NVAFX_STATUS_MODEL_LOAD_FAILED

The specified model file cannot be loaded.

NVAFX_STATUS_GPU_UNSUPPORTED

The GPU is unsupported. Audio effects SDK requires Turing or later GPU with Tensor cores

NVAFX_STATUS_GPU_UNSUPPORTED

The selected GPU is not supported. The SDK requires Turing and above GPU with Tensor cores.

NVAFX_STATUS_NO_SUPPORTED_GPU_FOUND

No supported GPU found on the system.

NVAFX_STATUS_WRONG_GPU

Current GPU is not the one selected.

Notice

The information provided in this specification is believed to be accurate and reliable as of the date provided. However, NVIDIA Corporation ("NVIDIA") does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This publication supersedes and replaces all other specifications for the product that may have been previously supplied.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and other changes to this specification, at any time and/or to discontinue any product or service without notice. Customer should obtain the latest relevant specification before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer. NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this specification.

NVIDIA products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on these specifications will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this specification. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this specification, or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this specification. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this specification is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, RTX™, and Turing are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com